

# Faster and Simpler Width-Independent Parallel Algorithms for Positive Semidefinite Programming

Richard Peng      Kanat Tangwongsan  
*Carnegie Mellon University\**

## Abstract

This paper studies the problem of finding a  $(1 + \varepsilon)$ -approximation to positive semidefinite programs. These are semidefinite programs in which all matrices in the constraints and objective are positive semidefinite and all scalars are non-negative. Previous work (Jain, Yao, *FOCS'11*) gave an NC algorithm that requires at least  $\Omega(\frac{1}{\varepsilon^{13}} \log^{13} n)$  iterations. The algorithm performs at least  $\Omega(n^\omega)$  work per iteration, where  $n$  is the dimension of the matrices involved, since each iteration involves computing spectral decomposition.

We present a simpler NC parallel algorithm that requires  $O(\frac{1}{\varepsilon^4} \log^4 n \log(\frac{1}{\varepsilon}))$  iterations. Moreover, if the positive SDP is provided in a factorized form, the total work of our algorithm can be bounded by  $\tilde{O}(n + q)$ , where  $q$  is the total number of nonzero entries in the factorization. Our algorithm is a generalization of Young's algorithm and analysis techniques for positive linear programs (Young, *FOCS'01*) to the semidefinite programming setting.

---

\*Computer Science Department. 5000 Forbes Ave. Pittsburgh, PA 15213. E-mail: {yangp,ktangwon}@cs.cmu.edu

# 1 Introduction

Semidefinite programming (SDP), alongside of linear programming, has been an important tool in approximation algorithms, optimization, and discrete mathematics. Very often, the goal is to efficiently find an approximation to the optimal solution of a given program. But like linear programs, SDPs are in general P-complete to solve exactly or to even approximate to any constant accuracy, suggesting that it is unlikely that there would be an NC or RNC algorithm for it. For this reason, we focus on parallel algorithms for finding approximate solutions to a particular class of semidefinite programs, known as positive SDPs [JY11]. These positive SDPs are a natural generalization of positive linear programs, which have been well studied in both sequential and parallel contexts (see, e.g., [LN93, PST95, GK98, You01, KY07, KY09]).

A semidefinite program, in general, can be solved to an arbitrary additive error  $\varepsilon$  using algorithms such as the Ellipsoid algorithm (with  $\text{poly}(1/\varepsilon)$  convergence) or the interior-point methods (with  $\text{poly}(\log \frac{1}{\varepsilon})$  convergence) [GLS93]. More recent efforts tend to focus on developing fast algorithms for finding a  $(1+\varepsilon)$ -approximation to SDPs, leading to a series of nice sequential algorithms (e.g., [AHK05, AK07]). These iterative algorithms all use multiplicative weight updates methods and the number of iterations required depends on the so-called “width” parameter of the input program. In some instances, this width parameter can be as large as  $\Omega(n)$ , making it the bottleneck in the depth of direct parallelization.

Most recently, Jain and Yao [JY11] studied an important class of SDPs known as positive SDPs and gave the first algorithm whose work and depth are independent of the width parameter (commonly known in the literature as width-independent algorithms). In this paper, we revisit their work and present a simpler and faster algorithm for solving positive SDPs to within a  $(1 + \varepsilon)$ -factor of the optimal solutions. The input consists of an accuracy parameter  $\varepsilon > 0$  and a positive semidefinite program (PSDP) in the following standard primal form:

$$\begin{aligned} \text{Minimize} \quad & \mathbf{C} \bullet \mathbf{Y} \\ \text{Subject to:} \quad & \mathbf{A}_i \bullet \mathbf{Y} \geq b_i \quad \text{for } i = 1, \dots, m \\ & \mathbf{Y} \succeq \mathbf{0}, \end{aligned} \tag{1.1}$$

where the matrices  $\mathbf{C}, \mathbf{A}_1, \dots, \mathbf{A}_m$  are  $n$ -by- $n$  symmetric positive semidefinite matrices, and the scalars  $b_1, \dots, b_m$  are non-negative reals. This is a subclass of SDPs, where in the general case, these matrices only have to be Hermitian and these scalars are only required to be real numbers. As with Jain and Yao, we assume that the input SDP has strong duality. The main theorem below quantifies the cost of our algorithm in terms of the number of iterations. The work and depth bounds implied by this theorem vary with the format of the input and how the matrix exponential are computed in each iteration. As we’ll discuss below, with input given in a suitable—and natural—form, our algorithm runs in nearly-linear work and polylogarithmic depth.

**Theorem 1.1 (Main Theorem)** *Given a primal positive SDP involving  $n \times n$  matrices with  $m$  constraints and an accuracy parameter  $\varepsilon > 0$ , there is an algorithm `approxPSDP` that produces a  $(1 + \varepsilon)$ -approximation in  $O(\frac{1}{\varepsilon^4} \log^4 n \log(\frac{1}{\varepsilon}))$  iterations, where each iteration involves only standard primitives on matrices and a special primitive that computes  $\exp(\Phi) \bullet \mathbf{A}$  ( $\Phi$  and  $\mathbf{A}$  are both positive semidefinite).*

## 1.1 Overview

Both Jain and Yao’s algorithm and our algorithm can be seen as a generalization of previous work on positive linear programs (positive LPs). Jain and Yao extended the algorithm of Luby and Nisan [LN93]. Their algorithm works directly on the primal program. Using the dual as guide, the update step is intricate as their analysis is based on carefully analyzing the eigenspaces of a particular matrix before and after each update. Unlike the algorithm of Jain and Yao, our solution relies, at the core, on an algorithm for solving the decision version of the dual program. We derive this core routine by generalizing the algorithm and analysis techniques of Young [You01], using the matrix multiplicative weights update (MMWU) mechanism in place of Young’s “soft” min and max. This leads to a very simple algorithm: besides standard operations on (sparse) matrix, the only special primitive needed is the matrix dot product  $\exp(\Phi) \bullet \mathbf{A}$ , where  $\Phi$  and  $\mathbf{A}$  are both positive semidefinite.

More specifically, our algorithm considers the following normalized primal/dual programs:

$$\begin{array}{c|c}
 \text{\textit{Primal}} & \text{\textit{Dual}} \\
 \hline
 \begin{array}{l} \text{Minimize} \quad \text{Tr}[\mathbf{Y}] \\ \text{Subject to:} \quad \mathbf{A}'_i \bullet \mathbf{Y} \geq 1 \quad \text{for } i = 1, \dots, m \\ \quad \quad \quad \mathbf{Y} \succeq 0 \end{array} & \begin{array}{l} \text{Maximize} \quad \mathbf{1}^\top \mathbf{x} \\ \text{Subject to:} \quad \sum_{i=1}^n x_i \mathbf{A}'_i \preceq \mathbf{I} \\ \quad \quad \quad \mathbf{x} \geq \mathbf{0}. \end{array}
 \end{array} \tag{1.2}$$

where  $\mathbf{I}$  represents the identity matrix. This is without loss of generality, as it can be obtained by “dividing through” by  $\mathbf{C}$  (see Appendix A). To solve this SDP, we design an algorithm for solving the decision version of it: Given a goal value  $\hat{\delta}$ , either find a solution  $\mathbf{x} \in \mathbb{R}_n^+$  to (1.2-D) with objective at most  $(1 + \varepsilon/2)\hat{\delta}$ , or solution  $\mathbf{Y}$  to (1.2-P) with objective at least  $(1 - \varepsilon/2)\hat{\delta}$ . By further scaling the  $\mathbf{A}_i$ ’s, it suffices to only consider the case where  $\hat{\delta} = 1$ . With this algorithm, the optimization version can be solved by binary searching on the objective a total of at most  $O(\log(\frac{n}{\varepsilon}))$  iterations.

**Work and Depth.** Similar to the sequential setting [AHK05, AK07], the majority of the cost of each iteration of our algorithm comes from computing matrix exponential, or more precisely the values of  $\mathbf{A}'_i \bullet \Phi$ , where  $\Phi$  is some PSD matrix. The cost of our algorithm therefore depends on how the input is specified. When the input is given prefactored—that is, the  $n$ -by- $n$  matrices  $\mathbf{A}_i$ ’s are given as  $\mathbf{A}_i = \mathbf{Q}_i \mathbf{Q}_i^\top$  and the matrix  $\mathbf{C}^{-1/2}$  is given, then Theorem 4.1 can be used to compute matrix exponential in  $O(\frac{1}{\varepsilon^3}(n + M) \log^2 M \log(1/\varepsilon))$  work and  $O(\frac{1}{\varepsilon} \log^2 M \log(1/\varepsilon))$  span, where  $M$  is the number of nonzero entries across  $\mathbf{Q}_i$ ’s and  $\mathbf{C}^{-1/2}$ . This is because the matrix  $\Phi$  that we exponentiate has  $\|\Phi\|_2 \leq O(K)$ , as shown in Lemma 3.7.

Therefore, as a corollary to the main theorem, we have the following cost bounds:

**Corollary 1.2** *The algorithm approxPSDP can be implemented in  $\tilde{O}(n + M)$  work and  $O(\log^{O(1)} M)$  depth.*

If the input program is not given in this form, we can add a preprocessing step that factors each  $\mathbf{A}_i$  into  $\mathbf{Q}_i \mathbf{Q}_i^\top$  since  $\mathbf{A}_i$  positive semidefinite. Often, these matrices have certain structure that makes it easier to factor (e.g., Laplacians). In general, this preprocessing requires at most  $O(n^4)$  work and  $O(\log^3 n)$  depth using standard parallel QR factorization [JáJ92]. Similarly, we can factor and invert  $\mathbf{C}$  in the same cost bound. We can often do better if  $\mathbf{C}$  has some structure.

## 2 Background and Notation

In this section, we review notation and facts which will prove useful later in the paper. Throughout this paper, we use the notation  $\tilde{O}(f(n))$  to mean  $O(f(n) \text{polylog}(f(n)))$ .

**Matrices and Positive Semidefinite Matrices.** Unless otherwise stated, we will deal with real symmetric matrices in  $\mathbb{R}^{n \times n}$ . A symmetric matrix  $\mathbf{A}$  is positive semidefinite, denoted by  $\mathbf{A} \succeq \mathbf{0}$  or  $\mathbf{0} \preceq \mathbf{A}$ , if for all  $\mathbf{x} \in \mathbb{R}^n$ ,  $\mathbf{x}^\top \mathbf{A} \mathbf{x} \geq 0$ . Equivalently, this means that all eigenvalues of  $\mathbf{A}$  are non-negative and the matrix  $\mathbf{A}$  can be written as

$$\mathbf{A} = \sum_i \lambda_i \mathbf{v}_i \mathbf{v}_i^\top,$$

where  $\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$  are the eigenvectors of  $\mathbf{A}$  with eigenvalues  $\lambda_1 \geq \dots \geq \lambda_n$ , respectively. We will write  $\lambda_1(\mathbf{A}), \lambda_2(\mathbf{A}), \dots, \lambda_n(\mathbf{A})$  to represent the eigenvalues of  $\mathbf{A}$  in decreasing order.

Notice that positive semidefiniteness induces a partial ordering on matrices. For this, we write  $\mathbf{A} \preceq \mathbf{B}$  if  $\mathbf{B} - \mathbf{A} \succeq \mathbf{0}$ .

The trace of a matrix  $\mathbf{A}$ , denoted  $\text{Tr}[\mathbf{A}]$ , is the sum of the matrix’s diagonal entries:  $\text{Tr}[\mathbf{A}] = \sum_i A_{i,i}$ . Alternatively, the trace of a matrix can be expressed as the sum of its eigenvalues, so  $\text{Tr}[\mathbf{A}] = \sum_i \lambda_i(\mathbf{A})$ .

Furthermore, we define

$$\mathbf{A} \bullet \mathbf{B} = \sum_{i,j} A_{i,j} B_{i,j} = \text{Tr}[\mathbf{A}\mathbf{B}].$$

It follows that  $\mathbf{A}$  is positive semidefinite if and only if  $\mathbf{A} \bullet \mathbf{B} \geq 0$  for all PSD  $\mathbf{B}$ .

**Matrix Exponential.** Given an  $n \times n$  symmetric positive semidefinite matrix  $\mathbf{A}$  and a function  $f: \mathbb{R} \rightarrow \mathbb{R}$ , we define

$$f(\mathbf{A}) = \sum_{i=1}^n f(\lambda_i) \mathbf{v}_i \mathbf{v}_i^\top,$$

where, again,  $\mathbf{v}_i$  is the eigenvector corresponding to the eigenvalue  $\lambda_i$ . It is not difficult to check that for  $\exp(\mathbf{A})$ , this definition coincides with  $\exp(\mathbf{A}) = \sum_{i \geq 0} \frac{1}{i!} \mathbf{A}^i$ .

Our algorithm relies on a matrix multiplicative weights algorithm, which can be summarized as follows. For a fixed  $\varepsilon_0 \leq \frac{1}{2}$  and  $\mathbf{W}^{(1)} = \mathbf{I}$ , we play a repeated “game” a number of times, where in iteration  $t = 1, 2, \dots$ , the following steps are performed:

1. Produce a probability matrix  $\mathbf{P}^{(t)} = \mathbf{W}^{(t)} / \text{Tr}[\mathbf{W}^{(t)}]$ ,
2. Incur a gain matrix  $\mathbf{M}^{(t)}$ , and
3. Update the weight matrix as  $\mathbf{W}^{(t+1)} = \exp(\varepsilon_0 \sum_{t' \leq t} \mathbf{M}^{(t')})$ .

Like in the standard setting of multiplicative weight algorithms, the gain matrix is chosen by an external party, possibly adversarially. In our algorithm, the gain matrix is chosen to reflect the step we make in the iteration. Arora and Kale [AK07] shows that this algorithm offers the following guarantees (restated for our setting):

**Theorem 2.1 ([AK07])** For  $\varepsilon_0 \leq \frac{1}{2}$ , if  $\mathbf{M}^{(t)}$ ’s are all PSD and  $\mathbf{M}^{(t)} \preceq \mathbf{I}$ , then after  $T$  iterations,

$$(1 + \varepsilon_0) \sum_{t=1}^T \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \geq \lambda_{\max} \left( \sum_{t=1}^T \mathbf{M}^{(t)} \right) - \frac{\ln n}{\varepsilon_0}. \quad (2.1)$$

### 3 Parallel Algorithm for Positive SDPs

In this section, we describe a parallel algorithm for solving the positive packing SDPs inspired by Young’s algorithm for positive LPs. By binary searching on the objective, the problem can be reduced to  $O(\log(\frac{n}{\varepsilon}))$  iterations of the following decision problem: Given a goal value  $\hat{\omega}$ , either find a solution  $\mathbf{x} \in \mathbb{R}_n^+$  to (1.2-D) with objective at most  $(1 + \varepsilon/2)\hat{\omega}$ , or solution  $\mathbf{Y}$  to (1.2-P) with objective at least  $(1 - \varepsilon/2)\hat{\omega}$ . By further scaling the  $\mathbf{A}_i$ ’s, it suffices to only consider the case where  $\hat{\omega} = 1$ .

The main theorem for this section is the following:

**Theorem 3.1** *There is an algorithm `decisionPSDP` that given a positive SDP, in  $O\left(\frac{\log^3 n}{\varepsilon^4}\right)$  iterations either outputs a solution to the simplified packing problem (1.2-D) with objective at most  $1 + O(\varepsilon)$ , or a solution to the simplified covering problem (1.2-P) with objective at least  $1 - O(\varepsilon)$ .*

Presented in Algorithm 3.1 is an algorithm satisfying this theorem. Before we go about proving the theorem, let us look at the algorithm more closely. Fix an accuracy parameter  $\varepsilon > 0$ . We set  $K$  to  $\frac{1}{\varepsilon}(1 + \ln n)$ . The reason for this setting of  $K$  is technical, but the motivation was so that we can absorb the  $\ln n$  term in Theorem 2.1 and account for the contribution of the starting solution  $\mathbf{x}^{(0)}$ .

The algorithm is a multiplicative weight updates algorithm, which proceeds in rounds. Initially, we work with the starting solution  $\mathbf{x}_i^{(0)} = \frac{1}{n} \mathbf{A}_i$ . This solution is chosen to be small enough that  $\sum_i \mathbf{x}_i^{(0)} \mathbf{A}_i \preceq \mathbf{I}$  but contains enough mass that subsequent updates, which is a multiple of the current solution, are guaranteed to make rapid

---

**Algorithm 3.1** Parallel Packing SDP algorithm

---

Let  $K = \frac{1}{\varepsilon}(1 + \ln n)$ .

Let  $\mathbf{x}_i^{(0)} = \frac{1}{n \cdot \text{Tr}[\mathbf{A}_i]}$ .

Initialize  $\Psi^{(0)} = \sum_{i=1}^n \mathbf{x}_i^{(0)} \mathbf{A}_i$ .

For  $t = 1, \dots$ , while  $\mathbf{1}^\top \mathbf{x} \leq K$

1. Let  $\mathbf{W}^{(t)} = \exp(\Psi^{(t-1)})$ .
  2. Let  $p$  be such that  $(1 + \varepsilon)^{p-1} < \text{Tr}[\mathbf{W}^{(t)}] \leq (1 + \varepsilon)^p$ .
  3. Let  $B^{(t)} = \{i \in [n] : \mathbf{W}^{(t)} \bullet \mathbf{A}_i \leq (1 + \varepsilon)^{p+1}\}$ .
  4. If  $B^{(t)}$  is empty, return “infeasible” and  $\mathbf{W}^{(t)}$ .
  5. Let  $\delta^{(t)} = \alpha \cdot \mathbf{x}_B^{(t-1)}$ , where  $\alpha = \min\{\varepsilon / \|\mathbf{x}_B^{(t-1)}\|_1, \varepsilon / (1 + 10\varepsilon)K\}$ .
  6. Update  $\mathbf{x}^{(t)} = \mathbf{x}^{(t-1)} + \delta^{(t)}$  and  $\Psi^{(t)} = \Psi^{(t-1)} + \sum_{i=1}^n \delta^{(t)} \mathbf{A}_i$
- 

progress. In each iteration, we compute  $\mathbf{W}^{(t)} = \exp(\Psi^{(t-1)})$ , where  $\Psi^{(t-1)} = \sum_i x_i^{(t-1)} \mathbf{A}_i$ . (For some intuitions for the  $\mathbf{W}^{(t)}$  matrix, we refer the reader to [AK07, Kal07].

The next two lines of the algorithm is responsible for identifying which coordinates of  $\mathbf{x}$  will be updated. For starters, it may help to think of these lines as follows: let  $\mathbf{P}^{(t)} = \mathbf{W}^{(t)} / \text{Tr}[\mathbf{W}^{(t)}]$ —and  $B^{(t)}$  be simply  $\{i \in [n] : \mathbf{P}^{(t)} \bullet \mathbf{A}_i \leq 1 + \varepsilon\}$ . The algorithm discretizes  $\text{Tr}[\mathbf{W}^{(t)}]$  to ensure certain monotonicity properties on  $B^{(t)}$ . As we show later on in Lemma 3.2, the set  $B^{(t)}$  cannot be empty unless the system is infeasible. The final steps of the algorithm updates the values of  $\mathbf{x}^{(t)}$  for coordinates in  $B^{(t)}$  multiplicatively.

### 3.1 Analysis

To bound the approximation guarantees and the cost of this algorithm, we will rely on the following lemmas and claims that bound the spectrum of  $\Psi^{(t)}$ . Before we begin, we will need some notation and definition. An easy induction gives that the quantities that we track across the iterations satisfy the following relationships:

$$\mathbf{x}^{(t)} = \mathbf{x}^{(0)} + \sum_{\tau=1}^t \delta^{(\tau)} \tag{3.1}$$

$$\mathbf{W}^{(t)} = \exp(\Psi^{(t-1)}) \tag{3.2}$$

$$\mathbf{P}^{(t)} \stackrel{\text{def}}{=} \mathbf{W}^{(t)} / \text{Tr}[\mathbf{W}^{(t)}] \tag{3.3}$$

$$\text{Tr}[\mathbf{P}^{(t)}] = \text{Tr}[\mathbf{W}^{(t)} / \text{Tr}[\mathbf{W}^{(t)}]] = \text{Tr}[\mathbf{W}^{(t)}] / \text{Tr}[\mathbf{W}^{(t)}] = 1 \tag{3.4}$$

$$\mathbf{M}^{(t)} \stackrel{\text{def}}{=} \frac{1}{\varepsilon} \sum_{i=1}^n \delta_i^{(t)} \mathbf{A}_i \quad \text{when } t \geq 0 \tag{3.5}$$

$$\Psi^{(t)} = \sum_{i=1}^n \mathbf{x}_i^{(t)} \mathbf{A}_i = \varepsilon \sum_{\tau=0}^t \mathbf{M}^{(\tau)} \tag{3.6}$$

First, we show that  $B^{(t)}$  can never be empty unless the system is infeasible:

**Lemma 3.2 (Feasibility)** *If there is an iteration  $t$  such that  $B^{(t)}$  is empty, then there does not exist  $\mathbf{x}$  such that*

1.  $\mathbf{1}^\top \mathbf{x} \geq (1 + \varepsilon)K$
2.  $\sum_{i=1}^n \mathbf{x}_i \mathbf{A}_i \preceq \mathbf{I}K$

*Proof:* Note first that  $\text{Tr} [\mathbf{P}^{(t)}] = 1$ . Furthermore, the fact that  $B^{(t)}$  is empty means that for all  $i = 1, \dots, n$ ,  $\mathbf{W}^{(t)} \bullet \mathbf{A}_i \geq (1 + \varepsilon)^{p+1}$ . But we know that  $\text{Tr} [\mathbf{W}^{(t)}] > (1 + \varepsilon)^{p-1}$ , so

$$\mathbf{P}^{(t)} \bullet \mathbf{A}_i = \frac{\mathbf{W}^{(t)}}{\text{Tr} [\mathbf{W}^{(t)}]} \bullet \mathbf{A}_i \geq (1 + \varepsilon)^2,$$

Therefore,  $\mathbf{P}^{(t)}$  is a valid dual solution with objective at most 1, which in turn means no primal solution with objective more than 1 exists.  $\blacksquare$

We now analyze the initial solution:

**Claim 3.3**

$$\lambda_{\max} (\Psi^{(0)}) = \lambda_{\max} \left( \sum_{i=1}^n x_i^{(0)} \mathbf{A}_i \right) \leq 1$$

*Proof:* Our choice of  $\mathbf{x}^{(0)}$  guarantees that for all  $i = 1, \dots, n$ ,

$$x_i^{(0)} \mathbf{A}_i = \frac{1}{n \text{Tr} [\mathbf{A}_i]} \mathbf{A}_i \preceq \frac{1}{n} \mathbf{I}.$$

Summing across  $i = 1, \dots, n$  gives the desired bound.  $\blacksquare$

Note that the claim implies that the starting solution satisfies  $\sum_i x_i^{(0)} \mathbf{A}_i \preceq \mathbf{I} \preceq \mathbf{IK}$ —but it may not yet be a feasible solution for the whole system because it does not satisfy  $\mathbf{1}^\top \mathbf{x}^{(0)} > K$ . In the following lemma, we bound the  $\ell_1$  norm of the solution vector at every intermediate step. Notice that the initial solution may not satisfy  $\|\mathbf{x}^{(0)}\|_1 = \mathbf{1}^\top \mathbf{x}^{(0)} \leq K + \varepsilon$ , but if this happens, the algorithm will stop right away and the initial solution is trivially a feasible solution (since already  $x_i^{(0)} \mathbf{A}_i \preceq \mathbf{IK}$ ).

**Lemma 3.4** *Let  $T$  be the iteration in which the algorithm terminates. For  $t = 1, \dots, T$ ,*

$$\mathbf{1}^\top \mathbf{x}^{(t)} \leq K + \varepsilon.$$

*Proof:* Since  $T$  is the iteration when the algorithm terminate and for all  $t \geq 0$ ,  $\delta^{(t)} \in \mathbb{R}_+^n$ , we have

$$\begin{aligned} \mathbf{1}^\top \mathbf{x}^{(t)} &\leq \mathbf{1}^\top \mathbf{x}^{(T-1)} + \mathbf{1}^\top \delta^{(T)} \\ &\leq K + \mathbf{1}^\top \delta^{(T)} \end{aligned}$$

Furthermore, we know that  $\mathbf{1}^\top \mathbf{x}^{(0)} = \sum_i 1/(n \text{Tr} [\mathbf{A}_i])$ . But by our choice of  $\alpha$ , we know that  $\alpha \leq \varepsilon / \|\mathbf{x}_B^{(t-1)}\|_1$  and so  $\mathbf{1}^\top \delta^{(T)} = \|\delta^{(T)}\|_1 \leq \varepsilon$ , so  $\mathbf{1}^\top \mathbf{x}^{(t)} \leq K + \varepsilon$ .  $\blacksquare$

**Spectrum Bound.** Finally, we bound the spectrum of our solutions:

**Lemma 3.5 (Spectrum Bound)** *For  $t = 0, \dots, T$ , where  $T$  is the final iteration,*

$$\Psi^{(t)} = \sum_{i=1}^n x_i^{(t)} \mathbf{A}_i \preceq (1 + 10\varepsilon) K \mathbf{I}. \quad (3.7)$$

We prove this lemma by resorting to Theorem 2.1, which relates the final spectral values to the “gain” derived at each intermediate step. For this, we show a claim that quantifies the gain we get in each step as a function of the  $\ell_1$ -norm of the change we make in that step.

**Claim 3.6** For all  $t = 1, \dots, T$ ,

$$\mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} \leq \frac{(1 + \varepsilon)^2}{\varepsilon} \cdot \|\delta^{(t)}\|_1. \quad (3.8)$$

*Proof:* Consider that

$$\begin{aligned} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} &= \frac{1}{\varepsilon} \left( \sum_{i=1}^n \delta_i^{(t)} \mathbf{A}_i \right) \bullet \mathbf{P}^{(t)} \\ &= \frac{1}{\varepsilon} \left( \sum_{i \in B} \delta_i^{(t)} \mathbf{A}_i \bullet \mathbf{P}^{(t)} \right) \end{aligned}$$

Every  $i \in B$  has the property that

$$\mathbf{A}_i \bullet \mathbf{W}^{(t)} \leq (1 + \varepsilon)^{p+1}$$

So then, since

$$\mathbf{P}^{(t)} = \frac{\mathbf{W}^{(t)}}{\text{Tr}[\mathbf{W}^{(t)}]} \preceq \frac{\mathbf{W}^{(t)}}{(1 + \varepsilon)^{p-1}},$$

we have

$$\mathbf{A}_i \bullet \mathbf{P}^{(t)} \leq (1 + \varepsilon)^2$$

and thus

$$\begin{aligned} \mathbf{M}^{(t)} \bullet \mathbf{P}^{(t)} &\leq \frac{1}{\varepsilon} \left( \sum_{i \in B} \delta_i^{(t)} (1 + \varepsilon)^2 \right) \\ &\leq \frac{(1 + \varepsilon)^2}{\varepsilon} \mathbf{1}^\top \delta^{(t)}. \end{aligned}$$

■

*Proof of Lemma 3.5:* We can rewrite  $\Psi^{(t)}$  as

$$\Psi^{(t)} = \sum_{i=1}^n x_i^{(0)} \mathbf{A}_i + \sum_{\tau=1}^t \sum_{i=1}^n \delta_i^{(\tau)} \mathbf{A}_i = \sum_{i=1}^n x_i^{(0)} \mathbf{A}_i + \varepsilon \sum_{\tau=1}^t \mathbf{M}^{(\tau)},$$

so

$$\lambda_{\max}(\Psi^{(t)}) \leq \lambda_{\max} \left( \sum_{i=1}^n x_i^{(0)} \mathbf{A}_i \right) + \varepsilon \cdot \lambda_{\max} \left( \sum_{\tau=1}^t \mathbf{M}^{(\tau)} \right)$$

since both sums yield positive semidefinite matrices.

By Claim 3.3, we know that the first term is at most 1. To bound the second term, we again apply Theorem 2.1, which we restate below:

$$(1 + \varepsilon) \sum_{\tau=1}^t \mathbf{M}^{(\tau)} \bullet \mathbf{P}^{(\tau)} \geq \lambda_{\max} \left( \sum_{\tau=1}^t \mathbf{M}^{(\tau)} \right) - \frac{\ln n}{\varepsilon}$$

Rearranging terms, we have

$$\lambda_{\max} \left( \sum_{\tau=1}^t \mathbf{M}^{(\tau)} \right) \leq (1 + \varepsilon) \sum_{\tau=1}^t \mathbf{M}^{(\tau)} \bullet \mathbf{P}^{(\tau)} + \frac{\ln n}{\varepsilon}.$$

We also want to make sure that each  $\mathbf{M}^{(\tau)}$  satisfies  $\mathbf{M}^{(\tau)} \preceq \mathbf{I}$ . With an easy induction on  $t$ , we can show that (3.7) holds for all  $\tau \leq t - 1$ . This means that for  $\tau = 1, \dots, t$ , each  $\mathbf{M}^{(\tau)}$  satisfies

$$\mathbf{M}^{(\tau)} = \frac{1}{\varepsilon} \sum_{i=1}^n \delta_i^{(\tau)} \mathbf{A}_i \preceq \frac{\alpha}{\varepsilon} \sum_{i=1}^n x_i^{(\tau)} \mathbf{A}_i \preceq \frac{\varepsilon/(1+10\varepsilon)K}{\varepsilon} \sum_{i=1}^n x_i^{(\tau)} \mathbf{A}_i \preceq \frac{\varepsilon/(1+10\varepsilon)K}{\varepsilon} (1+10\varepsilon)K \mathbf{I} \preceq \mathbf{I},$$

since  $\alpha \leq \varepsilon/(1+10\varepsilon)K$ . It then follow from Claim 3.6 that

$$\varepsilon \cdot \lambda_{\max} \left( \sum_{\tau=1}^t \mathbf{M}^{(\tau)} \right) \leq (1+\varepsilon) \sum_{\tau=1}^t (1+\varepsilon)^2 \cdot \mathbf{1}^\top \delta^{(\tau)} + \ln n,$$

which means

$$\varepsilon \cdot \lambda_{\max} \left( \sum_{\tau=1}^t \mathbf{M}^{(\tau)} \right) \leq \ln n + (1+\varepsilon)^4 K$$

because  $\mathbf{1}^\top \delta^{(t)} \leq K + \varepsilon \leq (1+\varepsilon)K$ .

Combining these altogether, we get

$$\lambda_{\max}(\Psi^{(t)}) \leq 1 + \ln n + (1+\varepsilon)^4 K \leq \varepsilon K + (1+\varepsilon)^4 K \leq (1+10\varepsilon)K.$$

■

## 3.2 Cost

Similar to Young's analysis, our analysis relies on the notion of phases, grouping together iterations with similar  $\mathbf{W}^{(t)}$  matrices into a phase in a way that ensures the existence of a coordinate  $i$  within it with the property that this coordinate is incremented (i.e.,  $\delta_i^{(t)} > 0$ ) by a significant amount in every iteration of this phase. To this end, we say that an iteration  $t$  belongs to *phase*  $p$  if and only if  $(1+\varepsilon)^{p-1} < \text{Tr}[\mathbf{W}^{(t)}] \leq (1+\varepsilon)^p$ . A phase ends when the algorithm terminates or the next iteration belongs to a different phase.

Almost immediate from this definition is a bound on the number of phases:

**Lemma 3.7** *The total number of phases is at most  $O(K/\varepsilon)$ .*

*Proof:* On the one hand, we have  $\mathbf{0} \preceq \Psi^{(0)}$ , so  $\text{Tr}[\mathbf{W}^{(0)}] \geq n \cdot e^0 = n$ . On the other hand, we know that  $\Psi^{(T)} \preceq (1+O(\varepsilon))K$ , so  $\text{Tr}[\mathbf{W}^{(t)}] \leq n \exp((1+O(\varepsilon))K)$ . This means that the total of number is phases is at most

$$\log_{1+\varepsilon} \frac{\text{Tr}[\mathbf{W}^{(T)}]}{\text{Tr}[\mathbf{W}^{(0)}]} \leq \log_{1+\varepsilon} \exp((1+O(\varepsilon))K) \quad (3.9)$$

$$\leq \frac{1}{\varepsilon} (1+O(\varepsilon))K = O\left(\frac{K}{\varepsilon}\right) \quad (3.10)$$

■

To bound the total number of steps, we'll analyze the number of iterations within a phase. For this, we'll need a couple of claims. The first claim shows that if a coordinate is incremented at the end of a phase, it must have been incremented in every iteration of this phase.

**Claim 3.8** *If  $i \in B^{(t)}$ , then for all  $t' < t$  belonging to the same phase as  $t$ ,  $i \in B^{(t')}$ .*



*Proof:* Suppose  $t$  belongs to phase  $p$ . As  $i \in B^{(t)}$ , we know that  $\mathbf{W}^{(t)} \bullet \mathbf{A}_i \leq (1 + \varepsilon)^{p+1}$ . Since  $t' < t$  we have

$$\Psi^{(t)} - \Psi^{(t')} = \sum_{t' \leq \tau < t} \mathbf{M}^{(\tau)} \quad (3.11)$$

$$= \sum_{t' \leq \tau < t} \sum_i \delta_i^{(\tau)} \mathbf{A}_i \succeq 0 \quad (3.12)$$

Therefore  $\mathbf{W}^{(t')} \preceq \mathbf{W}^{(t)}$  and that

$$\mathbf{W}^{(t')} \bullet \mathbf{A}_i \leq \mathbf{W}^{(t)} \bullet \mathbf{A}_i \leq (1 + \varepsilon)^{p+1} \quad (3.13)$$

Which means that  $i \in B^{(t')}$ , as desired.  $\blacksquare$

In the second claim, we'll place a bounding box around each coordinate  $x_i^{(t)}$  of our solution vectors. This turns out to be an important machinery in bounding the number of iterations required by the algorithm.

**Claim 3.9 (Bounding Box)** *For all index  $i$ , at any iteration  $t$ ,*

$$x_i^{(t)} \leq (1 + O(\varepsilon))n^2 K / \text{Tr}[\mathbf{A}_i] x_i^{(0)}$$

*Proof:* Recall that our initial solution sets  $x_i^{(0)} = 1/(n \text{Tr}[\mathbf{A}_i])$ . To argue an upper bound on  $x_i^{(t)}$ , note that Lemma 3.5 gives

$$\Psi^{(t)} \preceq (1 + O(\varepsilon))\mathbf{I}K \quad (3.14)$$

Since  $\sum_{j=1}^n x_j^{(t)} \mathbf{A}_j = \Psi^{(t)}$  and each  $\mathbf{A}_i$  is positive semidefinite, we have

$$\text{Tr}[x_i^{(t)} \mathbf{A}_i] \leq \text{Tr}[\Psi^{(t)}] \leq (1 + O(\varepsilon))nK \quad (3.15)$$

We conclude that  $x_i^{(t)} \leq (1 + O(\varepsilon))n^2 K / \text{Tr}[\mathbf{A}_i] x_i^{(0)}$ , as claimed.  $\blacksquare$

The final claim shows that each iteration makes significant progress in incrementing the solution.

**Claim 3.10** *In each iteration, either  $\mathbf{1}^\top \delta^{(t)} = \varepsilon$  or  $\alpha \geq \Omega(\varepsilon/K)$ .*

*Proof:* We chose  $\alpha$  to be  $\min\{\varepsilon/\|\mathbf{x}_B^{(t-1)}\|_1, \varepsilon/(1+10\varepsilon)K\}$ . If  $\alpha = \varepsilon/\|\mathbf{x}_B^{(t-1)}\|_1$ , then  $\|\delta^{(t)}\|_1 = \varepsilon$  and we are done. Otherwise, we have  $\alpha = \varepsilon/(1+10\varepsilon)K$ , which is  $\Omega(\varepsilon/K)$ .  $\blacksquare$

**Corollary 3.11** *The number of iterations per phase is at most*

$$O\left(\frac{K}{\varepsilon} \ln(nK)\right).$$

*and the total number of iterations is at most:*

$$O\left(\frac{\log^3 n}{\varepsilon^4}\right)$$

*Proof:* Consider a phase of the algorithm, and let  $f$  be the final iteration of this phase. By Claim 3.10, each iteration  $t$  causes  $\mathbf{1}^\top \delta^{(t)} = \varepsilon$  or  $\alpha \geq \Omega(\varepsilon/K)$ . Since  $\mathbf{1}^\top \mathbf{x}^{(t)} \leq K + \varepsilon$  for all  $t \leq T$ , the number of iterations in which  $\mathbf{1}^\top \delta^{(t)} = \varepsilon$  can be at most  $O(K/\varepsilon)$ . Now, let  $i \in B^{(f)}$  be a coordinate that got incremented in the final iteration of this phase. By Claim 3.8, this coordinate got incremented in every iteration of this phase. Therefore, the number of iterations within this phase in which  $\alpha \geq \Omega(\varepsilon/K)$  is at most

$$\log_{1+\Omega(\varepsilon/K)}(x_i^{(f)}/x_i^{(0)}) \leq \log_{1+\Omega(\varepsilon/K)}((1 + O(\varepsilon))n^2 K) = O\left(\frac{K}{\varepsilon} \ln((1 + O(\varepsilon))K)\right). \quad (3.16)$$

Combining with Lemma 3.7 and the setting of  $K = O\left(\frac{\log n}{\varepsilon}\right)$  gives the overall bound.  $\blacksquare$

## 4 Evaluation of $\exp(\Phi) \bullet \mathbf{A}_i$

We show the following result about computing the matrix dot product of a positive semidefinite matrix and the matrix exponential of another positive semidefinite matrix.

**Theorem 4.1** *Suppose  $\Phi$  has  $p$  non-zero entries,  $\|\Phi\|_2 \leq \kappa$ , and the  $n$ -by- $n$  matrices  $\mathbf{A}_i$ 's are given as  $\mathbf{A}_i = \mathbf{Q}_i \mathbf{Q}_i^\top$ , where the total number of nonzeros across  $\mathbf{Q}_i$ 's is  $q$ . Then, there is an algorithm `bigDotExp`( $\Phi, \{\mathbf{A}_i = \mathbf{Q}_i \mathbf{Q}_i^\top\}_{i=1}^n$ ) that computes  $\exp(\Phi) \bullet \mathbf{A}_i$  upto a factor- $(1 \pm \varepsilon)$  approximation in  $O(\kappa \log n \log(1/\varepsilon) + \log \log n)$  depth and  $O(\frac{1}{\varepsilon^2}(\kappa \log(1/\varepsilon)p + q) \log n)$  work.*

The idea behind proving Theorem 4.1 is to approximate the matrix exponential using a low degree polynomial, as evaluating matrix exponentials exactly is costly. For this, we will apply the following lemma, reproduced from Lemma 6 in [AK07]:

**Lemma 4.2** ([AK07]) *If  $\mathbf{B}$  is a PSD matrix such that  $\|\mathbf{B}\|_2 \leq \kappa$ , then the operator*

$$\hat{\mathbf{B}} = \sum_{0 \leq i < k} \frac{1}{i!} \mathbf{B}^i \quad \text{where } k = \max\{e^2 \kappa, \ln(2\varepsilon^{-1})\}$$

*satisfies*

$$(1 - \varepsilon) \exp(\mathbf{B}) \preceq \hat{\mathbf{B}} \preceq \exp(\mathbf{B}).$$

*Proof of Theorem 4.1:* The given factorization of each  $\mathbf{A}_i$  allows us to write  $\exp(\Phi) \bullet \mathbf{A}_i$  as the 2-norm of a vector:

$$\exp(\Phi) \bullet \mathbf{A}_i = \text{Tr} \left[ \exp(\Phi) \mathbf{Q}_i \mathbf{Q}_i^\top \right] \tag{4.1}$$

$$= \text{Tr} \left[ \mathbf{Q}_i^\top \exp\left(\frac{1}{2}\Phi\right) \exp\left(\frac{1}{2}\Phi\right) \mathbf{Q}_i \right] \tag{4.2}$$

$$= \|\exp\left(\frac{1}{2}\Phi\right) \mathbf{Q}_i\|_2^2 \tag{4.3}$$

By Lemma 4.2, it suffices to evaluate  $\hat{\mathbf{B}} \bullet \mathbf{A}_i$  where  $\hat{\mathbf{B}}$  is an approximation to  $\mathbf{B} = \exp(\frac{1}{2}\Phi)$ . To further reduce the work, we can apply the Johnson-Lindenstrauss transformation to reduce the length of the vectors to  $O(\log n)$ ; specifically, we find a  $O(\frac{1}{\varepsilon^2} \log n) \times n$  Gaussian matrix  $\Pi$  and evaluate

$$\|\Pi \hat{\mathbf{B}} \mathbf{Q}_i\|_2 \tag{4.4}$$

Since  $\Pi$  only has  $O(\frac{1}{\varepsilon^2} \log n)$  rows, we can compute  $\Pi \hat{\mathbf{B}}$  using  $O(\log n)$  evaluations of  $\hat{\mathbf{B}}$ . The work/depth bounds follow from doing each of the evaluations of  $\hat{\mathbf{B}} \Pi_i$ , where  $\Pi_i$  denotes the  $i$ -th column of  $\Pi$ , and matrix-vector multiplies involving  $\Phi$  in parallel. ■

## References

- [AHK05] Sanjeev Arora, Elad Hazan, and Satyen Kale. Fast algorithms for approximate semidefinite programming using the multiplicative weights update method. In *FOCS*, pages 339–348, 2005. 1, 2
- [AK07] Sanjeev Arora and Satyen Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007. 1, 2, 3, 4, 9
- [GK98] Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proceedings of the 39th Symposium on the Foundations of Computer Science (FOCS)*, pages 300–309, 1998. 1

- [GLS93] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, New York, 2nd edition, 1993. [1](#)
- [JáJ92] Joseph JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992. [2](#)
- [JY11] Rahul Jain and Penghui Yao. A parallel approximation algorithm for positive semidefinite programming. In *FOCS*, pages 463–471, 2011. [1](#), [10](#)
- [Kal07] Satyen Kale. *Efficient Algorithms using the Multiplicative Weights Update Method*. PhD thesis, Princeton University, August 2007. Princeton Tech Report TR-804-07. [4](#)
- [KY07] Christos Koufogiannakis and Neal E. Young. Beating simplex for fractional packing and covering linear programs. In *FOCS*, pages 494–504, 2007. [1](#)
- [KY09] Christos Koufogiannakis and Neal E. Young. Distributed and parallel algorithms for weighted vertex cover and other covering problems. In *PODC*, pages 171–179, 2009. [1](#)
- [LN93] Michael Luby and Noam Nisan. A parallel approximation algorithm for positive linear programming. In *STOC’93*, pages 448–457, New York, NY, USA, 1993. [1](#)
- [PST95] Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Fast approximation algorithms for fractional packing and covering problems. *Math. Oper. Res.*, 20(2):257–301, 1995. [1](#)
- [You01] Neal E. Young. Sequential and parallel algorithms for mixed packing and covering. In *FOCS*, pages 538–546, 2001. [1](#)

## A Normalized Positive SDPs

This is the same transformation that Jain and Yao presented [[JY11](#)]; we only present it here for easy reference.

Consider the primal program in [\(1.1\)](#). It suffices to show that it can be transformed into the following program without changing the optimal value:

$$\begin{aligned}
 &\text{Minimize} && \text{Tr} [\mathbf{Z}] \\
 &\text{Subject to:} && \mathbf{B}_i \bullet \mathbf{Z} \geq 1 \quad \text{for } i = 1, \dots, m \\
 &&& \mathbf{Z} \succeq \mathbf{0},
 \end{aligned} \tag{A.1}$$

We can make the following assumptions without loss of generality: First,  $b_i > 0$  for all  $i = 1, \dots, m$  because if  $b_i$  were 0, we could have thrown it away. Second, all  $\mathbf{A}_i$ ’s are in the support of  $\mathbf{C}$ , or otherwise we know that the corresponding dual variable must be set to 0 and therefore can be removed right away. Therefore, we will treat  $\mathbf{C}$  as having a full-rank, allowing us to define

$$\mathbf{B}_i \stackrel{\text{def}}{=} \frac{1}{b_i} \mathbf{C}^{-1/2} \mathbf{A}_i \mathbf{C}^{-1/2}$$

It is not hard to verify that the normalized program [\(A.1\)](#) has the same optimal value as the original SDP [\(1.1\)](#).

Note that if we’re given factorization of  $\mathbf{A}_i$  into  $\mathbf{Q}_i \mathbf{Q}_i^\top$ , then  $\mathbf{B}_i$  can also be factorized as:

$$\mathbf{B}_i = \frac{1}{b_i} (\mathbf{C}^{-1/2} \mathbf{Q}_i) (\mathbf{C}^{-1/2} \mathbf{Q}_i)^\top$$

Futhermore, it can be checked that the dual of the normalized program is the same as the dual in Equation [1.2](#).

## B Proofs about Matrix Multiplicative Weights Update Algorithm

We give a proof of Theorem [2.1](#) for completeness. First, we state a fact which is easy to verify:

**Fact B.1** *If  $x$  is a scalar such that  $0 \leq x \leq \varepsilon$ , then:*

$$1 + x \leq e^x \leq 1 + (1 + 2\varepsilon)x$$

*and this also generalizes to P.S.D. matrices  $\mathbf{A}$  when  $0 \preceq \mathbf{A} \preceq \varepsilon \mathbf{I}$ :*

$$\mathbf{I} + \mathbf{A} \leq \exp \mathbf{A} \leq \mathbf{I} + (1 + 2\varepsilon)\mathbf{A}$$

Since in general  $\mathbf{AB} \neq \mathbf{BA}$ , we cannot expect  $\exp(\mathbf{A} + \mathbf{B}) = \exp(\mathbf{A}) \exp(\mathbf{B})$ . But we can relate their trace values using Golden-Thompson inequality:

**Lemma B.2 (Golden-Thompson inequality)** *If  $\mathbf{A}$  and  $\mathbf{B}$  are PSD matrices, then*

$$\text{Tr} [\exp(\mathbf{A} + \mathbf{B})] \leq \text{Tr} [\exp(\mathbf{A}) \exp(\mathbf{B})]$$

**Proof of Theorem 2.1:**

$$\text{Tr} [\mathbf{W}^{(t+1)}] = \text{Tr} [\exp(\mathbf{\Psi}^{(t)})] \tag{B.1}$$

$$= \text{Tr} [\exp(\mathbf{\Psi}^{(t-1)} + \varepsilon \mathbf{M}^{(t)})] \tag{B.2}$$

$$\leq \text{Tr} [\exp(\mathbf{\Psi}^{(t-1)}) \exp(\varepsilon \mathbf{M}^{(t)})] \tag{B.3}$$

$$= \text{Tr} [\mathbf{W}^{(t)} \exp(\varepsilon \mathbf{M}^{(t)})] \tag{B.4}$$

$$\leq \text{Tr} [\mathbf{W}^{(t)} (\mathbf{I} + \varepsilon(1 + 2\varepsilon) \mathbf{M}^{(t)})] \tag{B.5}$$

$$= \text{Tr} [\mathbf{W}^{(t)}] + \varepsilon(1 + 2\varepsilon) \mathbf{W}^{(t)} \bullet \mathbf{M}^{(t)} \tag{B.6}$$

$$= \text{Tr} [\mathbf{W}^{(t)}] (1 + \varepsilon(1 + 2\varepsilon) \mathbf{P}^{(t)} \bullet \mathbf{M}^{(t)}) \tag{B.7}$$

$$\leq \text{Tr} [\mathbf{W}^{(t)}] \exp(\varepsilon(1 + 2\varepsilon) \mathbf{P}^{(t)} \bullet \mathbf{M}^{(t)}) \tag{B.8}$$

$$\tag{B.9}$$

An easy induction with the base case that  $\text{Tr} [\mathbf{W}^0] = \text{Tr} [\mathbf{I}] = n$  then gives:

$$\text{Tr} [\exp(\mathbf{\Psi}^{(T)})] \geq n \exp\left(\sum_{t=1}^T \varepsilon(1 + 2\varepsilon) \mathbf{P}^{(t)} \bullet \mathbf{M}^{(t)}\right) \tag{B.10}$$

Using the fact that  $\text{Tr} [\exp(\mathbf{A})] \geq \exp(\lambda_{\max}(\mathbf{A}))$  and taking logs of both sides gives:

$$\lambda_{\max}(\mathbf{\Psi}^{(T)}) \geq \ln n + \sum_{t=1}^T \varepsilon(1 + 2\varepsilon) \mathbf{P}^{(t)} \bullet \mathbf{M}^{(t)} \tag{B.11}$$

Dividing both sides by  $\varepsilon$  and substituting in  $\mathbf{\Psi}^{(T)}) = \sum_{t=1}^T \mathbf{M}^{(t)}$  gives the desired result. ■